```
import numpy as np
import matplotlib.pyplot as plt
from astropy.io import fits
from scipy import stats
import re


import os
import pathlib
import glob

os.listdir()
```

```
['pixel_flux_dist_8.png',
 'frame_averages_mean.csv',
 'pixel_flux_dist_51.png',
 'pixel_flux_dist_7.png',
 'pixel_flux_dist_123.png',
 'dark-median.ipynb',
 'dark_mean.ipynb',
 'hot_pixel_percentages.csv',
 'sigma_vs_mean2.png',
 'pixel_flux_dist_40.png',
 'frame_averages_median.csv',
 'pixel_flux_dist_83.png',
 'pixel_flux_dist_141.png',
 'sigma_vs_mean1.png',
 'pixel_flux_dist_154.png',
 'pixel_flux_dist_94.png',
 'pixel_flux_dist_77.png',
 'dark.ipynb',
 'dark',
 'msc_practical_demo_ccd_char.ipynb',
 'ccdcharacter.zip',
 'pixel_flux_dist_121.png',
 'spatial_new.ipynb',
 'temporal_plot.ipynb',
 'pixel_flux_dist_38.png',
 'comparision.csv',
 'corrected_extinction-checkpoint.ipynb',
 'pixel_flux_dist_87.png',
 'pixel_flux_dist_10.png',
 'pixel_flux_dist_66.png',
 'sigma_vs_meanhalf.png',
 'hot_pixel_percentages_mean.csv',
 'sigma_vs_mean3.png',
 '.ipynb_checkpoints',
 'hot_pixel_percentages_median.csv',
 'temporal_new.ipynb',
 'pixel_flux_dist_133.png',
```

```
 'pixel_flux_dist_21.png',
 'flatfield',
 'pixel_flux_dist_98.png',
 'pixel_flux_dist_122.png',
 'pixel_flux_dist_174.png']
```

```
import glob
import re

# Load unsorted filenames
filenames =
glob.glob("/home/puja/Downloads/18_4/linearity/more/*.FIT")

# Custom sort: extract the **last number** in filename for sorting
def extract_number(f):
    match = re.search(r'(\d+)(?=\.FIT$)', f)
    return int(match.group(1)) if match else -1

# Sort based on that number
filenum = sorted(filenames, key=extract_number)

filenum
```

```
['/home/puja/Downloads/18_4/linearity/more/1.FIT',
 '/home/puja/Downloads/18_4/linearity/more/2.FIT',
 '/home/puja/Downloads/18_4/linearity/more/3.FIT',
 '/home/puja/Downloads/18_4/linearity/more/4.FIT',
 '/home/puja/Downloads/18_4/linearity/more/5.FIT',
 '/home/puja/Downloads/18_4/linearity/more/6.FIT',
 '/home/puja/Downloads/18_4/linearity/more/7.FIT',
 '/home/puja/Downloads/18_4/linearity/more/8.FIT',
 '/home/puja/Downloads/18_4/linearity/more/9.FIT',
 '/home/puja/Downloads/18_4/linearity/more/10.FIT']
```

```
Gain = 2.58             # e- per ADU
sigma_threshold = 4.0   # Adjust threshold for hot pixels . Should not
be too aggresive otherwise exposure correction would be affected!
binsize = 1


# Read all files, stack, and apply hot-pixel mask
stacked_img = np.array([fits.getdata(f) - 100 for f in filenum])    #
read all files and correct for the PEDESTEL (check header)
stacked_img = stacked_img * Gain                                    #
Convert AUDs to electrons

stacked_img.shape
```

```
(10, 510, 765)
```

```
# Identify hot pixels in each frame
```

```
median_frame = np.median(stacked_img, axis=(1, 2), keepdims=True)    #
Per-frame mean
std_frame = np.std(stacked_img, axis=(1, 2), keepdims=True)      #
Per-frame std dev
mean_frame_3d = np.array([np.full(stacked_img.shape[1:], val) for val
in mean_frame])

std_frame_3d = np.array([np.full(stacked_img.shape[1:], val) for val
in std_frame])



# masking

hot_pixel_mask = stacked_img > (median_frame + sigma_threshold *
std_frame)  # Find hot pixels
# Replace hot pixels with the mean value of the corresponding frame
# Replace the hot pixels with the corresponding mean value for each
frame using 3d_mean_frame
stacked_img_filtered = np.where(hot_pixel_mask, mean_frame_3d,
stacked_img)

print("Filtered Image:")
print(stacked_img_filtered)
```

```
Filtered Image:
[[[ 8826.18        8599.14        8570.76       ...  8624.94
    8601.72        8753.94      ]
  [ 8655.9         8601.72        8483.04       ...  8769.42
    8777.16        8735.88      ]
  [ 8699.76        8733.3         8769.42       ...  8550.12
    8676.54        8841.66      ]
  ...
  [ 8911.32        8707.5         8480.46       ...  8764.26
    8661.06        8591.4       ]
  [ 8738.46        8743.62        8741.04       ...  8849.4
    8769.42        8862.3       ]
  [ 8679.12        8880.36        8614.62       ...  8826.18
    8867.46        8815.86      ]]

 [[17363.4        17327.28       17203.44       ... 17322.12
   17500.14       17396.94      ]
  [17605.92       17654.94       17216.34       ... 17531.1
   17391.78       12062.82682691]
  [17593.02       17510.46       17236.98       ... 17314.38
   17489.82       17515.62      ]
  ...
  [17531.1        17440.8        17084.76       ... 17309.22
   17466.6        17275.68      ]
  [17484.66       17335.02       17120.88       ... 17613.66
```

```
   17680.74       17520.78      ]
  [17466.6        12062.82682691 17554.32       ... 17590.44
   12062.82682691 17515.62      ]]

 [[25978.02       25957.38       25730.34       ... 25908.36
   25913.52       26112.18      ]
  [26383.08       26223.12       25910.94       ... 17961.90560277
   17961.90560277 26316.        ]
  [26140.56       25959.96       25748.4        ... 25895.46
   26135.4        26104.44      ]
  ...
  [25523.94       25970.28       25993.5        ... 25931.58
   26094.12       26109.6       ]
  [26078.64       26269.56       26086.38       ... 26318.58
   26083.8        26179.26      ]
  [17961.90560277 26354.7        26058.         ... 17961.90560277
   26199.9        26254.08      ]]

 ...

 [[69169.8        68947.92       68738.94       ... 68947.92
   69667.74       69701.28      ]
  [69933.48       69866.4        69239.46       ... 69737.4
   46702.36827759 46702.36827759]
  [46702.36827759 46702.36827759 68965.98       ... 69149.16
   69386.52       46702.36827759]
  ...
  [69995.4        69796.74       69340.08       ... 69303.96
   69494.88       69871.56      ]
  [69672.9        69850.92       69737.4        ... 46702.36827759
   69874.14       46702.36827759]
  [46702.36827759 46702.36827759 46702.36827759 ... 46702.36827759
   46702.36827759 46702.36827759]]

 [[77743.14       76889.16       77214.24       ... 77676.06
   77423.22       77913.42      ]
  [77975.34       77866.98       77743.14       ... 78119.82
   78202.38       52359.1893584 ]
  [77598.66       77619.3        77523.84       ... 77260.68
   78132.72       52359.1893584 ]
  ...
  [78184.32       77696.7        77185.86       ... 77320.02
   77833.44       78233.34      ]
  [52359.1893584  77923.74       77833.44       ... 52359.1893584
   77954.7        52359.1893584 ]
  [52359.1893584  52359.1893584  52359.1893584  ... 52359.1893584
   52359.1893584  52359.1893584 ]]

 [[85908.84       85741.14       84853.62       ... 85447.02
   85299.96       86295.84      ]
```

```
  [86050.74       86306.16       85578.6        ... 57871.81311834
   86120.4        57871.81311834]
  [86228.76       86231.34       85289.64       ... 85864.98
   85506.36       57871.81311834]
  ...
  [86177.16       85746.3        85098.72       ... 85645.68
   85993.98       86308.74      ]
  [57871.81311834 85803.06       86040.42       ... 57871.81311834
   85991.4        57871.81311834]
  [57871.81311834 57871.81311834 57871.81311834 ... 57871.81311834
   57871.81311834 57871.81311834]]]
```

```
# Print dimensions of stacked_img and stacked_img_filtered
print("Dimensions of stacked_img:", stacked_img.shape)
print("Dimensions of stacked_img_filtered:",
stacked_img_filtered.shape)
```

```
Dimensions of stacked_img: (10, 510, 765)
Dimensions of stacked_img_filtered: (10, 510, 765)
```

```
ntot_total = stacked_img_filtered.size  # Total number of pixels

# Count the number of hot pixels before replacing them
nhot_total = hot_pixel_mask.sum()  # Counts the number of True values
(hot pixels)
```

```
ntot_total
```

```
3901500
```

```
nhot_total
```

```
4547
```

```
# Example hot_pixel_mask before replacing hot pixels

# Count the number of hot pixels for each frame (z value)
nhot_framewise = [hot_pixel_mask[z].sum() for z in
range(hot_pixel_mask.shape[0])]

# Print the number of hot pixels for each z value (frame)
for z, nhot in enumerate(nhot_framewise):
    print(f"Number of hot pixels for frame z={z}: {nhot}")
```

```
Number of hot pixels for frame z=0: 37
Number of hot pixels for frame z=1: 134
Number of hot pixels for frame z=2: 279
Number of hot pixels for frame z=3: 345
Number of hot pixels for frame z=4: 433
Number of hot pixels for frame z=5: 533
Number of hot pixels for frame z=6: 603
```

```
Number of hot pixels for frame z=7: 695
Number of hot pixels for frame z=8: 732
Number of hot pixels for frame z=9: 756
```

```
# Calculate the total number of pixels for each frame (2D array)
ntot_framewise = [stacked_img_filtered[z].size for z in
range(stacked_img_filtered.shape[0])]

# Print the total number of pixels for each frame
for z, ntot in enumerate(ntot_framewise):
    print(f"Total number of pixels for frame z={z}: {ntot}")
```

```
Total number of pixels for frame z=0: 390150
Total number of pixels for frame z=1: 390150
Total number of pixels for frame z=2: 390150
Total number of pixels for frame z=3: 390150
Total number of pixels for frame z=4: 390150
Total number of pixels for frame z=5: 390150
Total number of pixels for frame z=6: 390150
Total number of pixels for frame z=7: 390150
Total number of pixels for frame z=8: 390150
Total number of pixels for frame z=9: 390150
```

```
# Loop through each frame (z = 1to z = 9)
for z in range(len(ntot_framewise)):  # or use
range(stacked_img_filtered.shape[0]) for frames count
    # Get the number of hot pixels (nhot) and total pixels (ntot) for
the current frame
    nhot = nhot_framewise[z]
    ntot = ntot_framewise[z]

    # Calculate the percentage of hot pixels
    percentage_hot_pixels = (nhot * 100) / ntot

    # Print the number of hot pixels and the percentage for the
current frame
    print(f"Frame z={z} - Number of masked pixels: {nhot}
({percentage_hot_pixels:5.3f}%)")
```

```
Frame z=0 - Number of masked pixels: 37 (0.009%)
Frame z=1 - Number of masked pixels: 134 (0.034%)
Frame z=2 - Number of masked pixels: 279 (0.072%)
Frame z=3 - Number of masked pixels: 345 (0.088%)
Frame z=4 - Number of masked pixels: 433 (0.111%)
Frame z=5 - Number of masked pixels: 533 (0.137%)
Frame z=6 - Number of masked pixels: 603 (0.155%)
Frame z=7 - Number of masked pixels: 695 (0.178%)
```

```
Frame z=8 - Number of masked pixels: 732 (0.188%)
Frame z=9 - Number of masked pixels: 756 (0.194%)
```

```
# Initialize an empty list to store the average of each frame
average_values = []

# Iterate over each frame (z-axis)
for z in range(stacked_img_filtered.shape[0]):
    # Calculate the average of the 2D frame (z-th slice) and store it
    frame_average = np.nanmean(stacked_img_filtered[z])
    average_values.append(frame_average)

# Convert the list to a 1D array
average_values = np.array(average_values)

# Print the result
print("1D array of averages for each z frame:")
print(average_values)
```

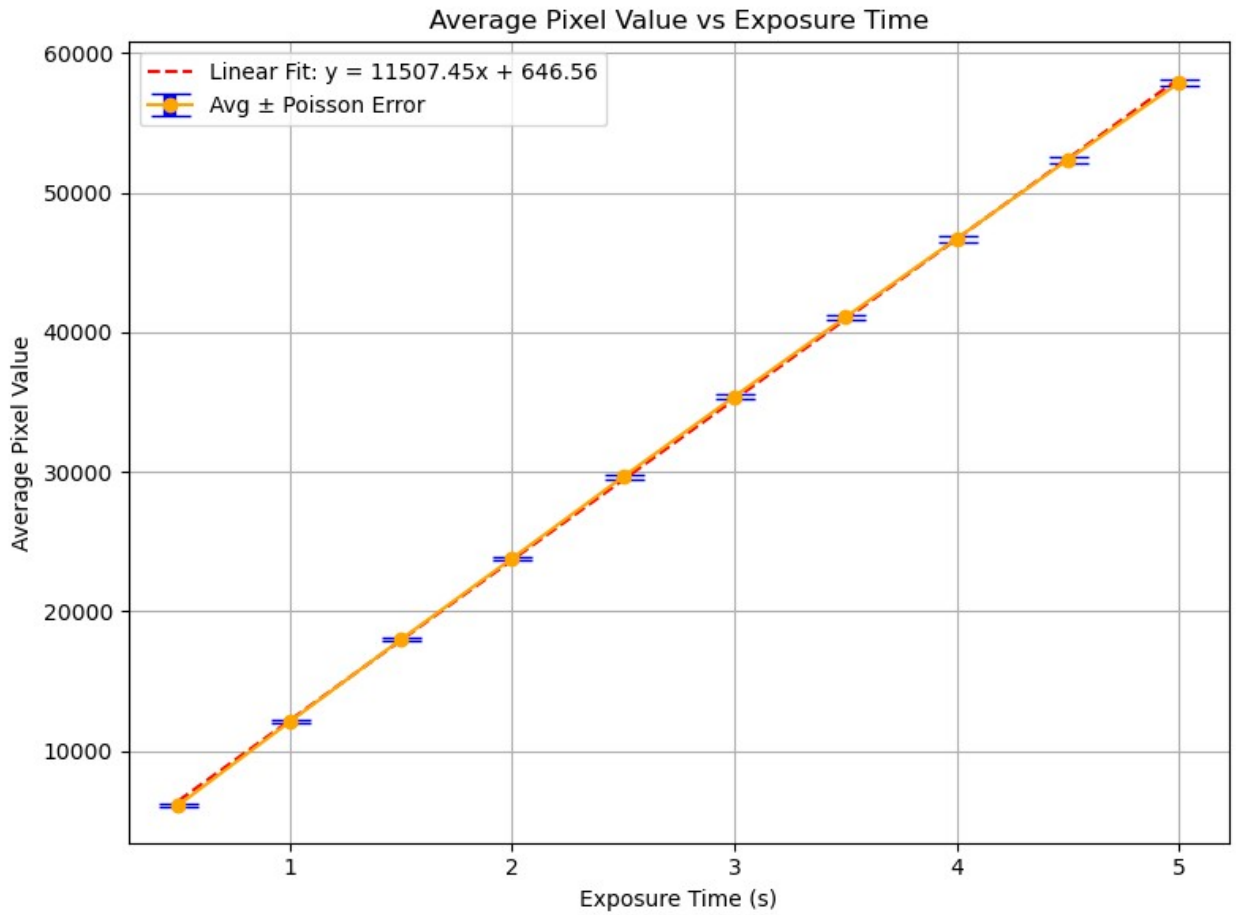
```
1D array of averages for each z frame:
[ 8505.88906581 17050.71020888 25563.52977179 34122.59943749
 42545.45356636 51121.02720753 59564.69421138 68223.11322818
 76330.76643001 84220.21904318]
```

```
# Initialize an empty list to store the average of each frame
average_values = []

# Iterate over each frame (z-axis)
for z in range(stacked_img_filtered.shape[0]):
    # Calculate the average of the 2D frame (z-th slice) and store it
    frame_average = np.nanmean(stacked_img_filtered[z])
    average_values.append(frame_average)

# Convert the list to a 1D array
average_values = np.array(average_values)

# Print the result vertically
print("1D array of averages for each z frame:")
for avg in average_values:
    print(avg)
```

```
1D array of averages for each z frame:
8505.889065807847
17050.710208880704
25563.529771788213
34122.59943748997
42545.45356635513
51121.02720752555
59564.69421138131
68223.11322817614
```

```
76330.76643001499
84220.21904318202

 #time=[0.5 1.0  1.5 2.0  2.5 3.0  3.5 4.0  4.5 5.0 ]

#LESS BRIGHTNESS
import numpy as np
import matplotlib.pyplot as plt

# Time in seconds
time = np.array([0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5])

# Average pixel values
average_values = np.array([
    6094.904847503069,
    12062.708283036356,
    17961.60953031101,
    23780.612965949,
    29632.939526963113,
    35383.53644885663,
    41077.30615624366,
    46700.62993558652,
    52357.038971075446,
    57869.177712739736 ])

# Poisson error
errors = np.sqrt(average_values)

# Linear fit
fit_params = np.polyfit(time, average_values, 1)  # degree 1
polynomial (line)
slope, intercept = fit_params
fit_line = np.poly1d(fit_params)

# Generate fit line points
time_fit = np.linspace(min(time), max(time), 100)
avg_fit = fit_line(time_fit)

# Plot
plt.figure(figsize=(8, 6))
plt.errorbar(time, average_values, yerr=errors, fmt='o-',
color='orange',
             ecolor='blue', elinewidth=6.0, capsize=9, label='Avg ±
Poisson Error')
plt.plot(time_fit, avg_fit, 'r--', label=f'Linear Fit: y =
{slope:.2f}x + {intercept:.2f}')
plt.xlabel("Exposure Time (s)")
plt.ylabel("Average Pixel Value")
plt.title("Average Pixel Value vs Exposure Time")
plt.grid(True)
```

```
plt.legend()
plt.tight_layout()
plt.show()

# Print actual values and Poisson errors
print("Exposure Time (s) | Avg Pixel Value ± Poisson Error")
print("-" * 50)
for t, val, err in zip(time, average_values, errors):
    print(f"{t:>6.1f} s           | {val:.2f} ± {err:.2f}")

# Print slope and intercept of the fitted line
print("\nLinear Fit Equation:")
print(f"y = {slope:.2f} * x + {intercept:.2f}")
```

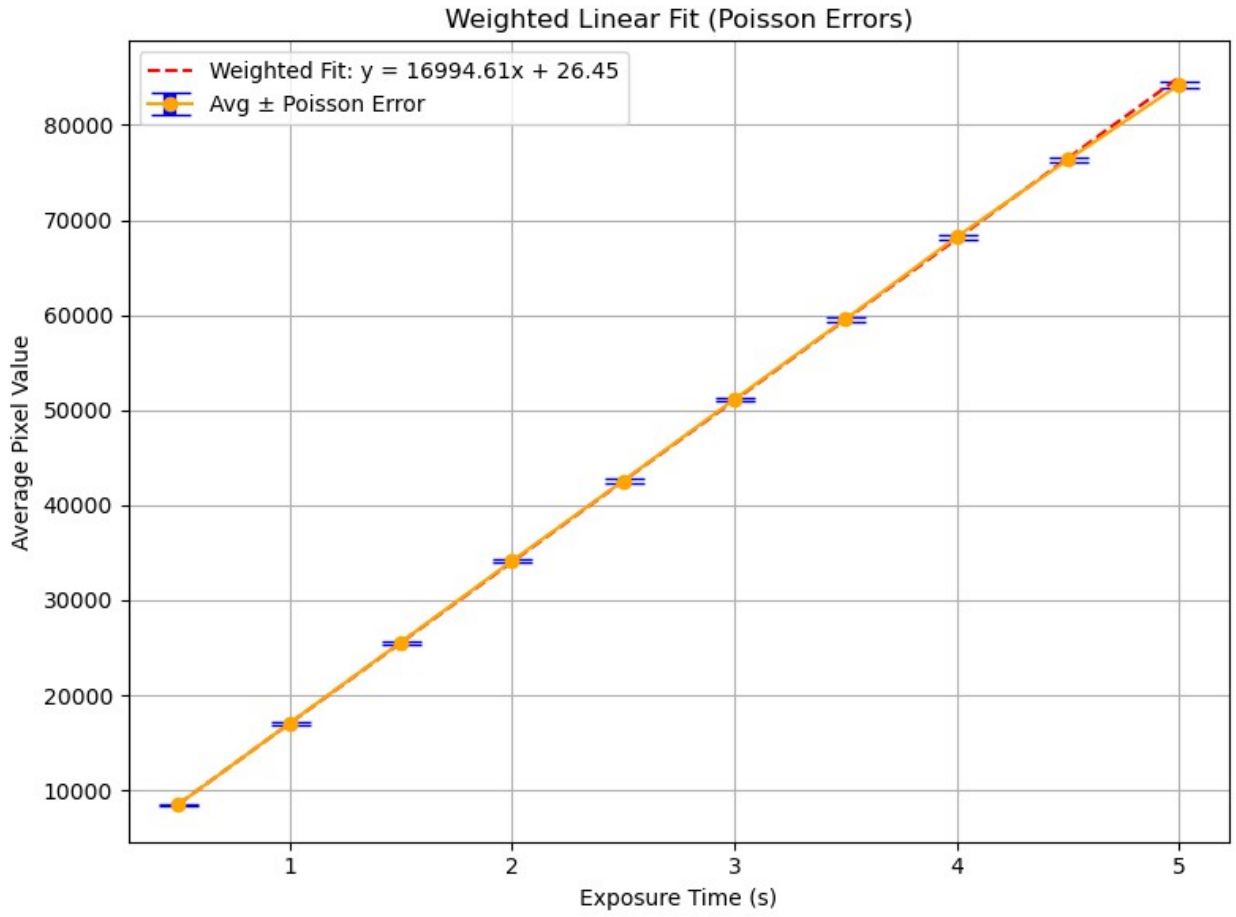



```
Exposure Time (s) | Avg Pixel Value ± Poisson Error
--------------------------------------------------
   0.5 s           | 6094.90 ± 78.07
   1.0 s           | 12062.71 ± 109.83
   1.5 s           | 17961.61 ± 134.02
   2.0 s           | 23780.61 ± 154.21
   2.5 s           | 29632.94 ± 172.14
```

```
   3.0 s           | 35383.54 ± 188.11
   3.5 s           | 41077.31 ± 202.68
   4.0 s           | 46700.63 ± 216.10
   4.5 s           | 52357.04 ± 228.82
   5.0 s           | 57869.18 ± 240.56

Linear Fit Equation:
y = 11507.45 * x + 646.56
```

```python
#MORE BRIGHTNESS
import numpy as np
import matplotlib.pyplot as plt

# Time in seconds
time = np.array([0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5])

# Brighter average pixel values
average_values = np.array([
    8505.889065807847,
    17050.710208880704,
    25563.529771788213,
    34122.59943748997,
    42545.45356635513,
    51121.02720752555,
    59564.69421138131,
    68223.11322817614,
    76330.76643001499,
    84220.21904318202
])

# Poisson errors (std dev)
errors = np.sqrt(average_values)

# Weights = 1 / variance = 1 / (std dev)^2 = 1 / average_value
weights = 1 / average_values

# Weighted linear fit
fit_params = np.polyfit(time, average_values, 1, w=weights)
slope, intercept = fit_params
fit_line = np.poly1d(fit_params)

# Generate fit line points
time_fit = np.linspace(min(time), max(time), 100)
avg_fit = fit_line(time_fit)

# Plot
plt.figure(figsize=(8, 6))
plt.errorbar(time, average_values, yerr=errors, fmt='o-',
color='orange',
             ecolor='blue', elinewidth=6.0, capsize=9, label='Avg ±
```

```
Poisson Error')
plt.plot(time_fit, avg_fit, 'r--', label=f'Weighted Fit: y =
{slope:.2f}x + {intercept:.2f}')
plt.xlabel("Exposure Time (s)")
plt.ylabel("Average Pixel Value")
plt.title("Weighted Linear Fit (Poisson Errors)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Print values and fit result
print("Exposure Time (s) | Avg Pixel Value ± Poisson Error")
print("-" * 50)
for t, val, err in zip(time, average_values, errors):
    print(f"{t:>6.1f} s          | {val:.2f} ± {err:.2f}")

print("\nWeighted Linear Fit Equation:")
print(f"y = {slope:.2f} * x + {intercept:.2f}")
```



```
Exposure Time (s) | Avg Pixel Value ± Poisson Error
--------------------------------------------------
   0.5 s          | 8505.89 ± 92.23
   1.0 s          | 17050.71 ± 130.58
   1.5 s          | 25563.53 ± 159.89
   2.0 s          | 34122.60 ± 184.72
   2.5 s          | 42545.45 ± 206.27
   3.0 s          | 51121.03 ± 226.10
   3.5 s          | 59564.69 ± 244.06
   4.0 s          | 68223.11 ± 261.20
   4.5 s          | 76330.77 ± 276.28
   5.0 s          | 84220.22 ± 290.21

Weighted Linear Fit Equation:
y = 16994.61 * x + 26.45
```